

MorphTagger v1.0 – User’s Manual

Roy Bar-Haim
Computer Science Department
Bar-Ilan University, Israel
barhair@cs.biu.ac.il
June 2, 2005

1. Introduction

MorphTagger is a part-of-speech tagger and word segmenter, based on HMM technology. **MorphTagger** was originally developed for Hebrew, but it is also suitable for other Semitic languages such as Arabic.

MorphTagger was developed in the Technion, by [Roy Bar-Haim](#), under the supervision of [Dr. Yoad Winter](#) (Technion) and [Dr. Khalil Sima’an](#) (University of Amsterdam).

The tagger receives a sequence of sentences with the possible morphological analyses for each word, generated by some morphological analyzer (see http://mila.cs.technion.ac.il/website/english/resources/morphological_analyzers/index for a list of public-domain morphological analyzers for Hebrew), and selects the most suitable analysis in context for each word. Each word analysis should be a sequence of morphemes, where each morpheme is tagged with a morpho-syntactic part-of speech. Except for this requirement, the tagger is completely language and tagset independent.

MorphTagger makes use of the following resources for training:

1. A manually tagged corpus.
2. (Optional) a probabilistic ambiguous corpus, that contains for each word its possible analyses. Such corpus can be automatically obtained by running a morphological analyzer on the raw text. The system assigns a probability for each possible analysis of each word token, and uses this corpus in conjunction with the manually tagged corpus, for better estimation of lexical probabilities - $P(\text{morpheme}|\text{POStag})$.

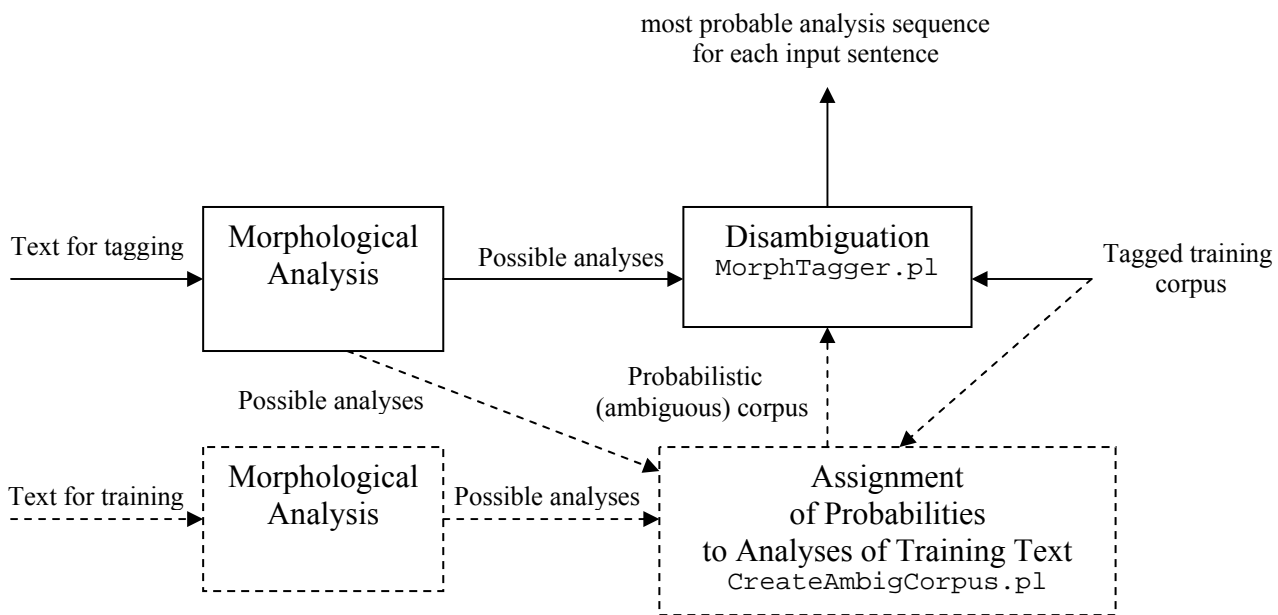
The disambiguation method is similar to standard HMM tagging. The main differences are:

- HMM modeling is done at the morpheme level, rather than at the word level. The morpheme-level model is translated to a word-level model, making it possible to use a standard HMM tagger for disambiguation. The translation is based on the observation that for a word in Hebrew, its segmentation into morphemes can be determined uniquely given the sequence of the POS tags of the morphemes (except for very rare cases).
- Usage of an ambiguous corpus for improving lexical probabilities, as explained above.

MorphTagger runs on UNIX and Windows (using CYGWIN). It makes use of the SRI Language Modeling Toolkit ([SRILM](#)). We are grateful to Andreas Stolcke of SRI International, for creating a lightweight version of SRILM, containing only the components that are necessary for MorphTagger.

The architecture of **MorphTagger** is illustrated in the diagram below.

For each component of MorphTagger, the corresponding program name is indicated.



2. License

MorphTagger is distributed under GNU Public License (GPL).

<http://www.gnu.org/copyleft/gpl.html>

The license **does not** include SRI Language Modeling Toolkit (SRILM), which is required by MorphTagger. For licensing information regarding SRILM, please refer to "Terms of Use" in SRILM website:

<http://www.speech.sri.com/projects/srilm/>

3. Installation

1. Prerequisites:
 - a. Perl (5.8.0 and above).
 - b. All the Prerequisites of SRILM (see below).

2. Install SRILM

SRILM can be downloaded from here:

<http://www.speech.sri.com/projects/srilm/download.html>

Either the full or the "Lite" version can be chosen. The "Lite" version contains only SRILM components that are needed by MorphTagger, and requires much smaller disk space.

3. Try `disambig -help` and `ngram-count -help` to make sure that the required SRILM components were installed.
4. Unpack `MorphTagger.tgz` (using `gunzip MorphTagger.tgz` and then `tar -xvf MorphTagger.tar`).
5. Add the full path of the directory to which you extracted the files to the search path variable (`PATH`) and to `PERL5LIB` environment variable. In addition, assign this path to an environment variable named `MORPHTAGGER`.
6. Test the tagger:
 - a. Download and unpack `SampleData.tgz`.
 - b. Enter the directory to which the sample data was saved and run:

```
MorphTagger.pl -ambig-corpus ambigcorpus.nf analyses.nf \  
corpus.nf taggerout
```

This should disambiguate the words in `analyses.nf` and the output should be written to `taggerout`.

- c. Compare the obtained output with the expected output:

```
diff taggerout expectedout
```

Files should be identical.

4. Input files

4.1 Types and formats of input files

Analyses file

Analyses file contains a list of words, followed by their possible analyses.

Format:

```
W11
    a111
    a112
    ...
W12
    a121
    a122
    ...
...
W1k
    a1k1
    a1k2
    ...
#
W21
    a211
    a212
    ...
W22
...
```

'#' marks end of sentence.

Each analysis is preceded by a tab. An analysis has the following format:

$(t_1 m_1) (t_2 m_2) \dots (t_k m_k)$

where the t_i 's are part-of-speech tags and the m_i 's are morphemes.

See file `analyses.nf` in the sample data files for example.

Ambiguous corpus file

This is a probabilistic corpus, typically generated by `CreateAmbigCorpus.pl`. Its format is almost identical to the analyses file. The only difference is that in the ambiguous corpus each analysis is preceded by a probability. The probabilities for the analyses for each word token should sum up to 1.

Corpus file

Contains analyzed (segmented and POS tagged) sentences.

Format:

```
{sentence #1}
W11 a11
W12 a12
...
W1k a1k
```

```
{sentence #2}
W21 a21
W22 a22
...
```

Each line contains a word – analysis pair. The format of analyses is described above.

4.2 Character set

Do not use the following characters for analyses, words and morphemes:

```
[ ] ( ) { } < > #
```

4.3 Sample data files

The following sample data files are included in SampleData.tgz:

Name	Description
corpus	sample training corpus
corpus.nf	sample training corpus, with features removed from the POS tags
analyses.nf	sample test analyses file to disambiguate, with features removed. analyses.nf and corpus.nf contain disjoint sentences
gold	The ``golden standard`` for the test analyses file. Given in the same format as MorphTagger's output
Expectedout	The expected output of morph tagger on analyses.nf, using corpus.nf and ambigcorpus.nf. Can be used to test MorphTagger's installation.
bigcorpus biccorpus.nf	A larger corpus (with/without features), extracted from the Hebrew Treebank. Subsumes both corpus(.nf) and analyses.nf
ambigcorpus.nf	Sample ambiguous corpus (without features).

The files are based on the Hebrew Treebank. See

<http://mila.cs.technion.ac.il/website/english/resources/corpora/treebank/index.html> for details about the treebank, the annotation scheme, etc.

5. Core Components

In this section we describe the language-independent components of MorphTagger, that perform the training and the disambiguation.

MorphTagger.pl

Trains the HMM model and disambiguates the given analyses file, one sentence at a time. In addition to the possible analyses in the analyses file, the tagger also considers analyses found in the corpus for the word. The output file contains the chosen analyses of the input words, one sentence per line. Each analysis is preceded by '[' and followed by ']'.

Usage: MorphTagger.pl [options] analysesfile corpusfile outputfile

analysesfile	analyses file to disambiguate.
corpusfile	Tagged training corpus
outputfile	output file for the tagging result.
Options	
-ambig-corpus	ambiguous probabilistic corpus. The ambiguous corpus is used to estimate lexical probabilities, which are interpolated with lexical probabilities estimated from the tagged corpus.
-lambda f	f should be between 0 and 1. Used for interpolation between lexical probabilities obtained from the tagged corpus and the ambiguous corpus: $\text{lambda} * P(\text{tagged corpus}) + (1 - \text{lambda}) * P(\text{ambiguous corpus})$ Default value:0.85 This option may be specified only in conjunction with -ambig-corpus or -bootstrap-input.
-input-to-ambig	Create an ambiguous corpus out of the input sentences, by invoking <code>CreateAmbiguousCorpus.pl</code> , which calculates the probability of each analysis for each word token in the input sentences. If another ambiguous corpus is specified by -ambig-corpus ,both ambiguous corpora are concatenated, and used in conjunction with the tagged corpus, as usual. Otherwise, only the corpus created from the input sentences is interpolated with the tagged corpus.
-workdir workdirname	Directory into which all the intermediate files will be written. If the directory does not exist, it will be created. The default value is <code>.MorphTaggerWorkDir</code> .
-help	Displays a 'Usage' message for the command

CreateAmbigCorpus.pl

Receives an analyses file and assigns probability for each analysis (including additional analyses for the input words, found in the tagged training corpus), thereby transforming the analyses file into a probabilistic ambiguous corpus. The probabilities are obtained by running a word-level HMM tagger, trained on the given tagged training corpus, and considering the 300 most probable sentence analyses.

Usage:

```
CreateAmbiguousCorpus.pl [options] analysesfile corpusfile outputfile
```

analysesfile	analyses file
corpusfile	tagged training corpus
outputfile	output file for the resulting probabilistic (ambiguous) corpus.
Options	
-workdir workdirname	Directory into which all the intermediate files will be written. If the directory does not exist, it will be created. The default value is <code>.MorphTaggerWorkDir</code> .
-help	Displays a 'Usage' message for the command

6. Integration with Segal's Hebrew Morphological Analyzer

In addition to the core components, MorphTagger package includes scripts that integrate with existing morphological analyzers for Hebrew. These scripts run the morphological analyzer on a given text, and translate its output to MorphTagger's format. The resulting file can be given as input to MorphTagger. Currently only Segal's analyzer is supported.

6.1 Installation

Segal's analyzer can be downloaded from

http://www.cs.technion.ac.il/~erelsgl/bxi/hmmtx/teud_tokna.html

The transliteration in which the input text should be provided is described in the site. Install the analyzer according to the instructions in the site.

In the directory where you want to run the analyzer add a symbolic link to 'hmmtx' directory of the analyzer (which contains the file `mcht.exe`), and name it 'Analyzer':

```
ln -fs analyzer-root/hmmtx Analyzer
```

Training the translator: the conversion from Segal's format to MorphTagger format has a trainable component for reducing overgeneration of analyses. The training results in two files in MorphTagger's directory: `rule.probs` and `addedrule.probs`. The files which are provided in MorphTagger's package were built from the "corpus" tagged corpus file. If a different file is used for training, these files may be regenerated by running

```
TrainTranslator.pl corpus
```

Where `corpus` is the tagged training corpus (with syntactic features).

6.2 Process overview

1. Initial preprocessing of the input text includes:
 - a. Sentence segmentation (each line in the text file should contain exactly one sentence).
 - b. Sentence tokenization – separate each token by white spaces. For example, the sentence

הח"כ אמר: "בוקר טוב!".

should look after tokenization like this:

הח"כ אמר : " בוקר טוב ! " .

Utilities for Hebrew sentence segmentation and tokenization can be found in the Knowledge Center for Processing Hebrew site:

<http://mila.cs.technion.ac.il/>

2. Convert the text to Segal's alphabet (documented in the analyzer's site) using *Transliterate.pl* (see below).
3. Run *Text2AnalysesSegal.pl* (see below) to create analyses file from the text file.

6.3 Programs

Transliterate.pl

Converts Hebrew text files between standard UTF-8 encoding and Segal's alphabet. The text is read from the standard input and written to the standard output. This utility was written by Shlomo Yona.

Usage: `Transliterate.pl [hebrew|segal]`

Options	
hebrew	The source text is in UTF-8 encoding
segal	The source text is in Segal's transliteration

Text2AnalysesSegal.pl

Usage: `Text2AnalysesSegal.pl [-f] textfile outputfile`

The script runs Segal's analyzer on the given `textfile`, converts the output to MorphTagger's format, and writes the result to `outputfile`.

Options	
-f	include syntactic features in the output. By default, the features are removed.